Graph Convolutional Network Hashing

Anonymous CVPR submission

Paper ID 1135

Abstract

Recently, graph-based hashing that learns similarity-preserving binary codes via an affinity graph has been intensively studied for large-scale image retrieval. However, most graph-based hashing methods resort to intractable binary quadratic programs, making them unscalable to massive data. In this paper, we propose a novel Graph Convolutional Network (GCN) based Hashing framework, dubbed GCNH, which directly carries out spectral convolution operations on both an image set and an affinity graph built over the set, naturally yielding a similarity-preserving binary embedding. GCNH fundamentally differs from conventional graph hashing methods which adopt an affinity graph as the only learning guidance in an objective function to pursue the binary embedding. As the core ingredient of GCNH, we introduce an intuitive Asymmetric Graph Convolutional (AGC) layer to simultaneously convolve the anchor graph, input data, and convolutional filters. By virtue of the AGC layer, GCNH well addresses the issues of scalability and out-of-sample extension when leveraging affinity graphs for hashing. As a use case of our GCNH, we particularly study the semi-supervised hashing scenario in this paper. Comprehensive evaluations on the CIFAR-10, NUS-WIDE and ImageNet datasets demonstrate the consistent advantages of GCNH over the state-of-the-art methods given limited labeled data.

1. Introduction

With the explosive growth of data nowadays, nearest neighbor search at scale has become a popular research subject. By encoding high-dimensional feature vectors into a set of compact binary codes, hashing has been recognized as an effective technique to accomplish large-scale similarity search, partially due to the extremely fast Hamming distance computation. Plenty of hashing techniques have been widely applied to deal with or facilitate practical problems, including approximate nearest neighbor search [11, 47, 19, 41], visual recognition [44, 24, 2, 37, 38], multi-modal analysis [10, 30], large-scale optimization [36], *etc*.

Generally speaking, learning to hash aims to pursue binary embeddings preserving the similarities among original data. The data similarities between data points are computed in a variety of ways. Supervised hashing methods [39, 26, 34, 27, 31, 15, 30, 5] define the similarities according to semantic labels, which have been shown to achieve promising performance. However, the heavy burden of collecting semantic labels hinders the applications of supervised hashing in practice. On the contrary, unsupervised hashing methods [12, 20, 16, 23, 9, 15] usually utilize pairwise distances, typically the Euclidean distances, between data points instead of labels to define the affinity graph. As an intersection between the supervised and unsupervised categories, semi-supervised hashing methods [46, 32] determine the similarities encompassing both labeled and unlabeled data. Recently, deep neural networks base hashing has been extensively studied [48, 21, 50, 22, 52, 29, 28, 8, 3]. Most of these methods form the key part of their hash functions by the Convolutional Neural Networks (CNNs) [18, 42, 45] model by taking advantage of its powerful representations.

Graph-based hashing has attracted intensive research attention in recent years [47, 35, 40, 33, 16, 23]. Graph-based hashing paradigms take an affinity graph, *i.e*., a pairwise similarity matrix, as the supervision in a properly designed objective, which typically results in an NP-hard *binary quadratic programming (BQP)* problem. As one of the most well-known graph-based hashing methods, Spectral Hashing (SH) [47] resolves an eigenfunction solution of 1-D graph Laplacians by holding an unrealistic assumption of uniform data distributions in all dimensions. To loosen this strong assumption, Anchor Graph Hashing (AGH) [35] adopts a small number of anchors to approximate the data similarities. However, the binary constraints still make the hashing problem difficult to solve. Then, most methods choose to discard the binary constraints and solve a relaxed continuous optimization problem, *e.g*., SH, AGH, and Inductive Manifold Hashing (IMH) [40]. However, the subsequent quantization step, which is usually done by thresholding real-valued embeddings to obtain binary codes, leads to inferior sub-optimal solutions. Lately,

the efforts [33] have been devoted to directly solving the original binary-constrained problem by casting graph-based hashing into a discrete optimization framework. Unfortunately, such methods still fall into the BQP problem, which requires heavy computational costs when encountering gigantic data. To this end, we aim to formulate a new graph hashing framework which efficiently scales to massive data without explicitly dealing with the hard optimization.

The recently proposed Graph Convolutional Networks (GCNs) [1, 25, 13, 17, 6] generalize CNNs to deal with non-Euclidean graph data items such as social networks, protein-interaction networks, and knowledge graphs. GCNs have been shown to be able to extract local, stationary, and compositional features from graphs. It is desirable to exploit a valid GCN to pursue a plausible binary embedding from an affinity graph constructed over a data set. To the best of our knowledge, our work is the first to introduce GCNs to learn binary codes. In order to efficiently apply GCNs, we should address two main difficulties: **(1)** Most previous works studied the graphs with only hundreds of nodes. Considering that the space complexity of a whole graph is $\mathcal{O}(n^2)$ with respect to n data points, GCNs can hardly scale to massive datasets, which hold as the main scenario of applying hashing techniques. **(2)** Another difficulty in leveraging GCNs to tackle hashing is the issue of out-of-sample extension. For an unseen query, one needs to quickly predict its hash code, while the current GCN models cannot handle an isolated data point without knowing its underlying affinity graph due to the transductive nature.

To tackle the aforementioned difficulties in applying GCNs to hashing, in this work, we introduce a novel Asymmetric Graph Convolutional (AGC) layer, which convolves the anchor graph [32], input images, and convolutional filters. Crucially, we incorporate the anchor graph into the AGC layer to leverage the similarities between original data points and selected anchor points. Hence, the space complexity of the similarity matrix is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(np)$ with respect to $p \ll n$ anchor points. Consequently, the AGC layer well addresses the scalability and out-of-sample extension issues which the GCN models in the literature suffered from.

Based on the above solution, in this work, we propose a novel hashing method, dubbed Graph Convolutional Network Hashing (GCNH), for learning binary codes given an affinity graph. Specifically, consisting of a stack of graph convolutional and fully connected layers, GCNH performs spectral convolution operations on input data points and their affinity graph to generate a binary embedding. Different from previous graph-based hashing methods only incorporating the graph into a binary code learning objective, our method takes both the data points and graph as the network inputs. This brings two-fold advantages. First, it avoids explicitly coping with the BQP problem. Second, by directly performing convolutions on the graph that captures data similarities, the proposed GCNH can yield an effective similarity-preserving binary embedding.

The GCNH framework is illustrated in Fig. 1, which stacks a graph convolutional layer, two AGC layers, and a fully connected layer. During the training stage, a set of data points and their affinity graph are fed to the GCN, and then the network outputs a low-dimensional binary embedding. While the GCNH framework can work with any affinity graph, we particularly focus on the semi-supervised hashing scenario, where the graph is constructed over both labeled and unlabeled data points. The combined unsupervised and supervised losses enable GCNH to seek compact and semantics-sensitive hash codes. The major contributions of this work include:

- We propose a novel Graph Convolutional Network Hashing (GCNH) approach, which can learn effective yet short similarity-preserving binary codes. Different from previous methods, the core graph convolutional layer of GCNH directly performs spectral convolution operations on the affinity graph, so that the generated low-dimensional binary embedding can naturally preserve the similarities among original data.

- We introduce a novel Asymmetric Graph Convolutional (AGC) layer, which convolves the anchor graph, input data, and convolutional filters. The AGC layer well addresses the scalability and out-of-sample extension issues of the previous GCN models when leveraging affinity graphs for hashing.

- We apply GCNH to the semi-supervised hashing scenario, and show that the resulting hashing mechanism is both scalable and capable of producing similarity-preserving binary codes given only a small number of labeled data points. The proposed GCNH demonstrates superior performance over the state-of-the-art hashing methods on several benchmark datasets, *e.g.*, NUS-WIDE and ImageNet.

The rest of the paper is organized as follows. We elaborate on the proposed GCNH approach in Section 2, including the core graph convolutional layer and asymmetric graph convolutional layer. Section 3 presents the experimental results, followed by the model analysis in Section 4. Finally Section 5 concludes the paper.

2. Graph Convolutional Network Hashing

The notations used throughout the paper are introduced as follows. We denote matrices as boldface uppercase letters like $\mathbf{X}$, vectors as boldface lowercase letters like $\mathbf{x}$, and scalars as lowercase letters like x. $\mathbf{I}_r$ denotes the $r \times r$ identity matrix, and vectors with all ones and zeros

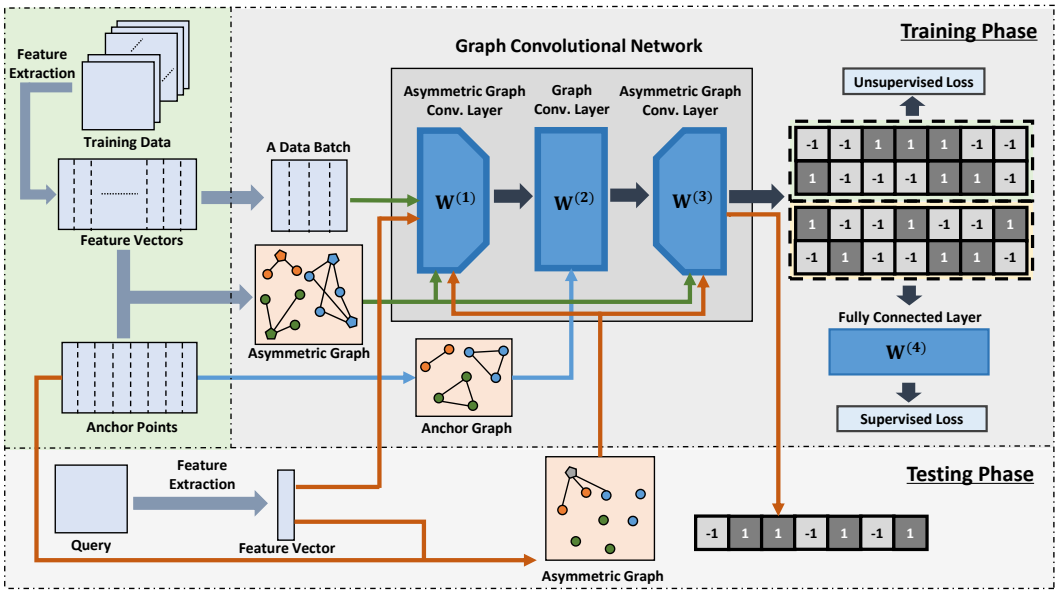

Figure 1: An illustration of the proposed GCNH approach. In the training phase, we sample a batch of data from the training set and randomly select some anchor points. The data batch, the asymmetric graph between sampled data and anchor points, and the anchor graph are simultaneously fed to the graph convolutional network (GCN). The output of the network is then quantized to generate binary codes, which are further regularized by the proposed supervised and unsupervised loss. During testing, a query and the corresponding asymmetric graph w.r.t. anchor points are fed to GCN to obtain the binary code (shown in orange line).

are denoted as $\mathbf{1}$ and $\mathbf{0}$, respectively. We abbreviate the Frobenius norm $\|\cdot\|_{\mathrm{F}}$ as $\|\cdot\|$ throughout this paper.

In the semi-supervised hashing scenario, we have $n = n_1 + n_2$ training data points, including n_1 labeled data points saved in rows of matrix $\mathbf{X}_l \in \mathbb{R}^{n_1 \times d}$ and n_2 unlabeled data points saved in rows of matrix $\mathbf{X}_u \in \mathbb{R}^{n_2 \times d}$. $\mathbf{Y} \in \{1, 0\}^{n_1 \times c}$ stores the corresponding labels of the labeled data $\mathbf{X}_l$, and each row of $\mathbf{Y}$ indicates the one-hot label for a data point from c classes. Without loss of generality, let $\mathbf{X} = [\mathbf{X}_l; \mathbf{X}_u] \in \mathbb{R}^{n \times d}$ have zero mean for all rows. The goal of semi-supervised hashing is to learn hash functions that map high-dimensional data vectors into compact semantics-sensitive binary codes.

2.1. Graph Convolutional Layer

The graph convolutional layers conduct spectral convolution operations on an affinity graph and input data points. In spectral graph theory, a spectral convolution on a graph is defined as the multiplication of a signal $\mathbf{s} \in \mathbb{R}^N$ with a filter $\mathbf{g}$ in the Fourier domain, *i.e.*, $\mathbf{g} * \mathbf{s} = \mathbf{U}\mathrm{diag}(\mathbf{g})\mathbf{U}^\top\mathbf{s}$, where $\mathbf{U}$ is the matrix collecting the eigenvectors of the graph Laplacian $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^\top$ (Λ includes the eigenvalues of $\mathbf{L}$ in its diagonal) and diag is the diagonal matrix operator. It is worth noting that the eigendecomposition of $\mathbf{L}$ is prohibitively expensive. To avoid this computational issue, Kipf and Welling [17] proposed a localized first-order approximation of the spectral graph convolution as:

$$\mathbf{H}^{(l)} = \sigma_l\big(\widetilde{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}\big), \tag{1}$$

where $\widetilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the normalized adjacency (similarity) matrix, $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$, and $\mathbf{A}$ is the original adjacency matrix. $\mathbf{W}^{(l)}$ behaves as the corresponding convolutional filters of the l^{th} layer. $\sigma_l(\cdot)$ denotes a properly chosen activation function. $\mathbf{H}^{(l-1)}$ and $\mathbf{H}^{(l)}$ are the input and output of the l^{th} layer, respectively. An in-depth discussion of this approximation can be found in [17].

A graph convolutional layer consists of three main components: $\mathbf{H}^{(l-1)}$ represents the features learned by the preceding layers, the convolutional filter $\mathbf{W}^{(l)}$ is supposed to learn more representative features, and the adjacency matrix $\widetilde{\mathbf{A}}$ serves as weighted summations. Specifically, for a node v_i, $\widetilde{\mathbf{A}}$ concatenates the features of v_i's neighboring nodes by a weighted summation and simultaneously assigns the new features to v_i, indicating that the features of neighboring nodes are encouraged to be closer.

2.2. Asymmetric Graph Convolutional Layer

The previous works using graph convolutional layers are under transductive settings. Obviously, it is impractical to directly employ GCN for large-scale tasks due to its high complexity and out-of-sample problems. To address the obstacles which hinder our applying GCNs to hashing, we propose an asymmetric variant of GCNs.

Specifically, instead of directly applying $\widetilde{\mathbf{A}}$ of $n \times n$, we form an asymmetric adjacency matrix $\widetilde{\mathbf{Z}}$ of $n \times p$ to present the similarities between data points and p randomly selected anchor points, *i.e.*, $\{\mathbf{a}_1, \cdots, \mathbf{a}_p\} \subset \mathbf{X}_l$. Here, $\widetilde{\mathbf{Z}}$ is the row-normalized matrix of the original asymmetric

adjacency matrix $\mathbf{Z}$ such that $\widetilde{\mathbf{Z}}\mathbf{1} = \mathbf{1}$. $\mathbf{Z}$ is defined as follows ($i = 1, \cdots, n;\ j = 1, \cdots, p$):

$$\mathbf{Z}_{ij} = \begin{cases} 1, & (\mathbf{x}_i, \mathbf{a}_j) \in \mathcal{M} \cup \mathbf{a}_j \in \mathcal{N}(\mathbf{x}_i), \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where $\mathcal{M}$ denotes the set in which $\mathbf{x}_i$ and $\mathbf{a}_j$ belong to the same class if $\mathbf{x}_i$ is a labeled point, and $\mathcal{N}(\mathbf{x}_i)$ denotes the set of s nearest anchor points of $\mathbf{x}_i$ if $\mathbf{x}_i$ is unlabled according to a certain distance metric[1].

Sketched in Fig. 1, we can write the spectral graph convolutions for the first Asymmetric Graph Convolutional (AGC) layer, the second Graph Convolutional (GC) layer, and the third AGC layer as follows:

$$\mathbf{H}^{(1)} = \sigma_1\big(\widetilde{\mathbf{Z}}^\top \mathbf{H}^{(0)} \mathbf{W}^{(1)}\big), \tag{3}$$

$$\mathbf{H}^{(2)} = \sigma_2\big(\widetilde{\mathbf{A}} \mathbf{H}^{(1)} \mathbf{W}^{(2)}\big), \tag{4}$$

$$\mathbf{H}^{(3)} = \sigma_3\big(\widetilde{\mathbf{Z}} \mathbf{H}^{(2)} \mathbf{W}^{(3)}\big), \tag{5}$$

in which $\mathbf{H}^{(0)}$ is always set to $\mathbf{X}$. In Eq. (4), $\widetilde{\mathbf{A}}$ is the normalized $\mathbf{A}$, where $\mathbf{A}_{ij} = 1$ if $(\mathbf{a}_i, \mathbf{a}_j) \in \mathcal{M}$ and 0 otherwise.

The first AGC layer (Eq. (3)) represents the anchors by the filtered features of its neighboring data points; the second GC layer (Eq. (4)) convolves the anchor representations $\mathbf{H}^{(1)}$ with the anchor graph and convolutional filter; the third AGC layer (Eq. (5)) learns features for each data point based on a weighted summation of its neighboring anchor representations $\mathbf{H}^{(2)}$, obtained from the preceding GC layer. By stacking these layers, the original data can be efficiently represented by the low-dimensional similarity-preserving embeddings, *i.e.*, the soft codes $\mathbf{U} = \mathbf{H}^{(3)}$. A simple quantization step is then applied to obtain the final binary codes $\mathbf{B}$, *i.e.*, $\mathbf{B} = \mathrm{sgn}(\mathbf{U})$, in which sgn is the elementwise sign function: $\mathrm{sgn}(x) = 1$ if $x > 0$, otherwise $\mathrm{sgn}(x) = -1$.

Scalability. Conventional symmetric adjacency matrices of size $n \times n$ will induce unacceptable memory load and computational costs for large-scale datasets with millions of data points. Owing to the proposed asymmetric adjacency matrix $\tilde{Z}$, we can reduce the space complexity significantly from $\mathcal{O}(n^2)$ to $\mathcal{O}(np)$, where $p \ll n$. During training, the relevant rows of $\widetilde{\mathbf{Z}}$ are selected to form an even smaller adjacency submatrix for each data batch, making our framework scalable to large-scale datasets. Moreover, by the AGC layer, we can naturally address the out-of-sample problem for unseen data. We will elaborate further on this point in Subsection 2.5.

2.3. Objective Function

Let the binary codes be $\mathbf{B} = [\mathbf{B}_l; \mathbf{B}_u]$, where $\mathbf{B}_l \in \{1, -1\}^{n_1 \times r}$ and $\mathbf{B}_u \in \{1, -1\}^{n_2 \times r}$ correspond to the r-bit binary codes of the labeled data $\mathbf{X}_l$ and unlabeled data $\mathbf{X}_u$, respectively. We expect the learned binary codes to preserve the original semantic/similarity structure. For this purpose, the binary codes of images in the same class or similar images in terms of the Euclidean distance should be as close as possible, while the binary codes of dissimilar images should be far away. Thus, the objective function naturally consists of two parts to deal with labeled and unlabeled images, respectively.

For the labeled images, we propose to minimize the classification error over them by considering the binary code learning as a linear classification problem, leading to the following objective function:

$$\begin{aligned} \min_{\mathbf{B}_l}\ & \mathcal{L}_1 := \big\|\mathbf{Y} - \mathbf{B}_l \mathbf{W}^{(4)}\big\|^2 \\ \text{s.t.}\ & \mathbf{B}_l \in \{1, -1\}^{n_1 \times r}, \end{aligned} \tag{6}$$

where $\mathbf{W}^{(4)} \in \mathbb{R}^{r \times c}$ retains the parameters of the fully connected layer (*i.e.*, the fourth layer of the GCNH model). We treat the binary codes $\mathbf{B}_l$ as low-dimensional binary features extracted by the preceding convolutional layers. The fully connected layer works as a classifier which classifies the binary codes to corresponding classes. This objective function $\mathcal{L}_1$ ensures the semantics-preserving property of the learned binary codes As such, images from the same class attain similar binary codes, while those from different classes have distinct codes.

For the unlabeled images, we propose to *softly* impose the uncorrelation and balance constraints on the bits of the binary codes $\mathbf{B}_u$ to maximize the information entropy:

$$\begin{aligned} \min_{\mathbf{B}_u}\ & \mathcal{L}_2 := \lambda\big\|\mathbf{B}_u^\top \mathbf{B}_u - n_2 \mathbf{I}_r\big\|^2 + \mu\big\|\mathbf{B}_u^\top \mathbf{1}\big\|^2 \\ \text{s.t.}\ & \mathbf{B}_u \in \{1, -1\}^{n_2 \times r}, \end{aligned} \tag{7}$$

where $\lambda, \mu > 0$ are trade-off parameters.

We combine the supervised classification loss $\mathcal{L}_1$ and the unsupervised loss $\mathcal{L}_2$ together to obtain the final objective function $\mathcal{L}$. Note that the optimization problem pertaining to this mixed objective is NP-hard due to the discrete constraints. In order to make the optimization tractable, we introduce an auxiliary variable $\mathbf{U} \in \mathbb{R}^{n \times r}$ to derive the following optimization problem (notice $\|\mathbf{B}_u^\top \mathbf{B}_u - n_2 \mathbf{I}_r\|^2 = \|\mathbf{B}_u^\top \mathbf{B}_u\|^2 + \text{const.}$):

$$\begin{aligned} \min_{\mathbf{U}, \mathbf{B}}\ & \mathcal{L} := \big\|\mathbf{Y} - \mathbf{U}_l \mathbf{W}^{(4)}\big\|^2 + \lambda\big\|\mathbf{U}_u^\top \mathbf{U}_u\big\|^2 + \mu\big\|\mathbf{U}_u^\top \mathbf{1}\big\|^2 \\ & \qquad + \gamma\|\mathbf{U} - \mathbf{B}\|^2 \\ \text{s.t.}\ & \mathbf{U} = [\mathbf{U}_l; \mathbf{U}_u] \in \mathbb{R}^{n \times r}, \mathbf{B} \in \{1, -1\}^{n \times r}, \end{aligned} \tag{8}$$

where the newly introduced loss $\mathcal{L}_3 := \gamma\|\mathbf{U} - \mathbf{B}\|^2$ ($\gamma > 0$) minimizes the quantization error between the continuous variables $\mathbf{U}$ and the binary codes $\mathbf{B}$.

The objective function $\mathcal{L}$ in Eq. (8) is what our proposed GCNH model tries to minimize. The partial derivatives of

[1] In this work, we choose the Euclidean distance.

Algorithm 1: GCNH Training

Input: Training data $\mathbf{X}$, p anchor points, number of nearest anchors s, and code length r.
Output: The parameters $\mathbf{W}^{(l)}$ of all layers of GCNH.
Initialization: Parameters $\mathbf{W}^{(l)}$ drawn from a Gaussian distribution, the adjacency matrix $\widetilde{\mathbf{A}}$ between anchor points, and the asymmetric adjacency matrix $\widetilde{\mathbf{Z}}$ between raw data points and anchor points.
repeat
1) Randomly sample a mini-batch $\mathbf{X}^b$ from both $\mathbf{X}_l$ and $\mathbf{X}_u$, and find the rows in $\widetilde{\mathbf{Z}}$ relevant to $\mathbf{X}^b$ to form an asymmetric adjacency submatrix;
2) $\mathbf{H}^{(0)} \leftarrow \mathbf{X}^b$, compute $\mathbf{U} \leftarrow \mathbf{H}^{(3)}$ by forward propagation, and obtain $\mathbf{B} = \mathrm{sgn}(\mathbf{U})$;
3) Compute the gradient with respect to $\mathbf{U}$ according to Eqs. (9)-(10);
4) Update the parameters $\mathbf{W}^{(l)}$ by back propagation using Eqs. (11)(5)(4)(3);
until *Converge*;

$\mathcal{L}$ with respect to three continuous variable matrices $\mathbf{U}_l$, $\mathbf{U}_u$ and $\mathbf{W}^{(4)}$ are given below:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}_l} = 2\big(\mathbf{U}_l\mathbf{W}^{(4)} - \mathbf{Y}\big)\big(\mathbf{W}^{(4)}\big)^\top + 2\gamma(\mathbf{U}_l - \mathbf{B}_l) \quad (9)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}_u} = 2\big(2\lambda\mathbf{U}_u\mathbf{U}_u^\top + \mu\mathbf{1}\mathbf{1}^\top\big)\mathbf{U}_u + 2\gamma(\mathbf{U}_u - \mathbf{B}_u) \quad (10)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(4)}} = 2\mathbf{U}_l^\top\big(\mathbf{U}_l\mathbf{W}^{(4)} - \mathbf{Y}\big). \quad (11)$$

Given an updated $\mathbf{U}$ (regarded as soft codes) we have $\mathbf{B} = \mathrm{sgn}(\mathbf{U})$. The training procedure of the proposed GCNH is summarized in Algorithm 1.

2.4. The GCNH architecture

We summarize the configuration of our GCNH model in Table 1, which contains two AGC layers, one GC layer, and one fully connected layer. In this paper, we use the four-layer architecture for a trade-off between accuracy and efficiency. The first and third layers are AGC layers, while the second layer is a GC layer. The activation functions for the first two layers are Rectification Linear Unit (ReLU), while the activation function for the third layer is hyperbolic tangent function (Tanh).

As illustrated in Fig. 1, a batch of data points and its corresponding asymmetric adjacency matrix are fed to the first AGC layer (gconv1), and convolved by the second GC layer (gconv2). The third AGC layer (gconv3) outputs the soft codes $\mathbf{U}$, which are finally quantized to binary codes $\mathbf{B}$ by thresholding at zeros.

Table 1: Configuration of the proposed GCNH model, where d denotes the dimension of features, r denotes the length of binary codes, and c denotes the number of classes.

Layer	Type	Configuration
gconv1	AGC layer	filter $d \times 2048$, ReLU
gconv2	GC layer	filter 2048×2048, ReLU
gconv3	AGC layer	filter $2048 \times r$, Tanh
fc4	Fully connected layer	$r \times c$

2.5. Out-of-Sample Extension

After the GCNH is sufficiently trained, one can use the network $(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)})$ to generate a binary code $\mathbf{b}_q \in \{1, -1\}^{1\times r}$ for any query image $\mathbf{q} \in \mathbb{R}^d$. In the inference process, an asymmetric adjacency matrix (single row) $\mathbf{Z}_q \in \mathbb{R}^{1\times p}$ is calculated according to the similarities between the query image $\mathbf{q}$ and the anchor points $\{\mathbf{a}_1, \cdots, \mathbf{a}_p\}$. Through forward propagation, the GCNH outputs a low-dimensional embedding as

$$\mathbf{u}_q = \sigma_3\left(\widetilde{\mathbf{Z}}_q\sigma_2\big(\widetilde{\mathbf{A}}\sigma_1\big(\widetilde{\mathbf{Z}}_q^\top\mathbf{q}^\top\mathbf{W}^{(1)}\big)\mathbf{W}^{(2)}\big)\mathbf{W}^{(3)}\right) \in \mathbb{R}^{1\times r}. \quad (12)$$

Finally, the binary code $\mathbf{b}_q$ of the query $\mathbf{q}$ is obtained by a simple quantization, *i.e.*, $\mathbf{b}_q = \mathrm{sgn}(\mathbf{u}_q)$.

3. Experiments

In this section, we evaluate the proposed method on three large-scale datasets, i.e., CIFAR-10[2], NUS-WIDE [4] and ImageNet [7], and compare it with several state-of-the-art hashing methods including non-deep and deep ones. In all experiments, data is normalized and zero-centered.

3.1. Datasets and Settings

CIFAR-10. The CIFAR-10 dataset consists of 60,000 color images from 10 classes (6,000 images per class). For each class, 100 images are uniformly sampled as the testing data, and the remaining images as the training data.

NUS-WIDE. The NUS-WIDE dataset contains about 270,000 images collected from Flickr, which are associated with 81 concept labels. Each image contains multiple semantic labels. As in [34], we collect the 21 most frequent labels to form the dataset of 193,755 images. For each label, 100 images are uniformly sampled for the testing set and the remaining images form the training set.

ImageNet. The large dataset ILSVRC 2012 is a subset of ImageNet, containing over 1.2 million images of totally 1,000 categories. We use the provided training set as our training set and 50,000 images from the validation set as the testing set.

Evaluation Metrics. In our experiments, we report the comparison results in terms of mean average precision

[2]https://www.cs.toronto.edu/ kriz/cifar.html

(mAP), mean precision of the top 500 retrieved neighbors (precision@500), Precision-Recall curves and Precision curves w.r.t different numbers of top returned images.

Implementation Details. We implement the proposed method based on the open source **PyTorch**[3] on a standard desktop PC with an NVIDIA 1080Ti GPU and 128G memory. In all experiments, the networks are trained by the stochastic gradient descent (SGD) with 0.9 momentum. For balancing the proportion of labeled data and unlabeled data in a batch, we empirically set the batch size to 120 with 40 labeled images and 80 unlabeled images. The weight decay is 5e-4. We set the parameters λ =1e-3, μ =1e-4 and γ =1e-4, respectively. Unless otherwise specified, 600 anchor points and 5 nearest anchors are selected for the (asymmetric) graph convolutional layers in all experiments.

3.2. Comparison with Non-Deep Hashing Methods

We compare our GCNH with 8 non-deep hashing methods, including semi-supervised S3PLH [46], supervised SDH [39], KSH [34] and FastH [26], and unsupervised ITQ [12], AGH [35] (with one or two layers) and DGH [33]. We denote AGH with one and two layers by AGH-1 and AGH-2, respectively. We use the implementations and suggested parameter settings provided by the authors. For SDH, KSH, AGH and DGH, 1,000 data points are randomly sampled as anchor points.

As for input features, we use 512-d GIST features for CIFAR-10, 500-d Bag-of-Words features for NUS-WIDE and 4,096-d deep features extracted from the activation of the last fully connected layer (FC7) of the pre-trained VGG16 network [42] for ImageNet.

For unsupervised hashing methods, all the available training data is used for training. As in [46], for semi-supervised hashing methods, we additionally select 1,000 images on CIFAR-10, 2,000 on NUS-WIDE and 20,000 on ImageNet randomly from the training set and assign them semantic label information. As for supervised hashing methods, only the above additionally selected images with semantic labels are used for training.

Table 2 and Table 3 show the mAP and precision@500 with 16, 32 and 64 bits on CIFAR-10, NUS-WIDE and ImageNet, respectively. In general, our method shows superiority over other methods with all code lengths. Particularly, our method achieves high performance with short code length, *e.g.*, ours obtains 32.23% mAP and 56.21% precision@500 with 16 bits on CIFAR-10, which are even much higher than the results of FastH with 64 bits (*i.e.*, 28.65% mAP and 38.48% precision@500).

The Precision-Recall and Precision curves with 32 bits on CIFAR-10 and NUS-WIDE are shown in Figs. 2 and 3, respectively. Clearly, our method consistently outperforms all other methods by large margins in all cases.

[3]http://www.pytorch.org

Table 2: Results in terms of mAP and mean precision of the top 500 retrieved neighbors (precision@500) on the CIFAR-10 dataset. Here, 512-d GIST features are used.

Methods	mAP			Precision@500		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
Ours	**31.98**	**32.47**	**33.22**	**56.21**	**56.34**	**56.86**
S3PLH	20.50	22.33	23.79	27.10	30.16	33.18
SDH	24.14	25.24	26.79	33.12	35.53	37.24
KSH	17.45	24.34	26.07	25.15	33.99	37.06
FastH	24.36	26.77	28.65	33.34	36.19	38.48
ITQ	17.06	17.64	18.21	26.03	27.69	29.41
AGH-1	16.73	15.89	14.87	28.24	29.91	30.68
AGH-2	17.18	17.25	16.72	27.30	29.79	31.17
DGH	16.63	16.44	16.19	23.35	23.96	23.50

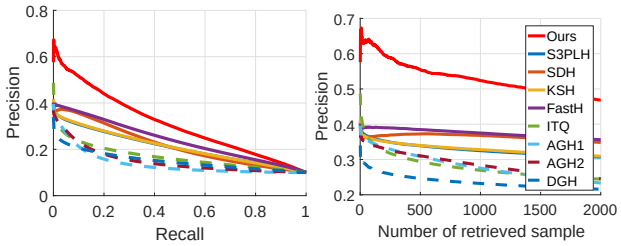

Figure 2: (Left) Precision-Recall and (right) Precision curves with top 2,000 retrieved images on CIFAR-10 with 32-bit codes.

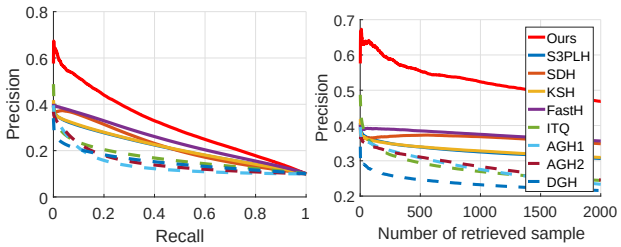

Figure 3: (Left) Precision-Recall and (right) Precision curves with top 2,000 retrieved images on NUS-WIDE with 32-bit codes.

For the large-scale ImageNet dataset, we report the mAP and precision@500 in Table 4. Obviously, our method outperforms other methods in mAP and precision@500 with all code lengths. Due to the extremely high computational cost of precision and recall, we do not illustrate the Precision-Recall and Precision curves on this dataset.

3.3. Comparison with Deep Hashing Methods

To further verify that our method can learn powerful binary codes from the similarity graph, we evaluate our method with five deep hashing methods: SSDH [49], DHN [51], DNNH [21], CNNH [48] and its variant CNNH⋆. In addition, we also compare our method with four non-deep hashing methods using deep features: S3PLH [46], SDH [39], KSH [34] and FastH [26].

In this experiment, we adopt CIFAR-10 and NUS-WIDE for evaluation. For fair comparison, we strictly follow the data-split strategies from [48]. The test sets are formed

Table 3: Results in terms of mAP (%) and mean precision (%) of the top 500 retrieved neighbors (precision@500) on the NUS-WIDE dataset. Here, 500-d Bag-of-Words features are used.

Methods	mAP			Precision@500		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
Ours	**44.62**	**48.79**	**49.03**	**57.03**	**59.16**	**60.15**
S3PLH	43.67	45.36	46.87	46.97	49.82	51.17
SDH	43.06	42.85	43.16	47.62	48.01	49.02
KSH	43.97	44.86	44.20	47.33	51.19	51.53
FastH	44.56	47.56	48.15	46.59	50.85	54.78
ITQ	39.34	38.89	39.27	45.35	47.04	48.19
AGH-1	38.61	38.42	38.05	42.74	39.38	43.62
AGH-2	37.59	38.09	38.27	40.62	45.41	47.26
DGH	37.01	36.60	36.53	43.61	43.89	43.90

(a) AlexNet Features (b) Binary Codes

Figure 4: t-SNE visualization of AlexNet features and binary codes of the randomly selected 5,000 data points from the training set of CIFAR-10. The code length of 32 bits is used.

by the same settings in section 3.1. On both datasets, we randomly sample 500 labeled images per class as the training set for the supervised methods, while for semi-supervised methods (*e.g.*, SSDH and ours), the rest training images are additionally used as their unlabeled data. In terms of image features, we use the pre-trained AlexNet [18] to extract 4,096-d deep features for fair comparison. According to the settings in [48], we evaluate all methods with binary codes of 12, 24, 32 and 48 bits.

Table 5 shows the mAP results on the CIFAR-10 and NUS-WIDE datasets. Obviously, our method keeps a large margin over non-deep methods with deep features in terms of all code lengths. Compared with the semi-supervised S3PLH, our method nearly doubles the mAP that S3PLH achieves on the CIFAR-10 dataset. Meanwhile, our method also outperforms the deep hashing methods. Specifically, our method achieves 80.4% mAP with 12 bits and outperforms all methods excluding SSDH by at least 18% with 48 bits. Further, we visualize the learned binary codes by t-SNE in Fig. 4. The AlexNet features naturally form separate clusters where the features in the same class are close to each other. The learned binary codes have smaller intra-class distances and larger inter-class distances, showing the effectiveness of learning compact semantics-sensitive and similarity-preserving binary codes.

Table 4: Results in terms of mAP (%) and mean precision (%) of the top 500 retrieved neighbors (precision@500) on the ImageNet dataset with 16, 32 and 64 bits, respectively. Here, the mAP scores are calculated based on the top 5,000 returned neighbors, and the 4,096-d deep VGG16 features are used.

Methods	mAP			Precision@500		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
Ours	**8.90**	**18.50**	**19.81**	**8.66**	**18.49**	**20.07**
S3PLH	6.84	14.15	17.63	7.26	15.04	18.28
SDH	7.71	15.37	17.99	7.94	16.41	18.93
KSH	7.36	15.10	17.74	7.69	16.19	18.56
FastH	7.68	15.76	18.34	8.07	16.94	19.17
ITQ	5.35	9.16	14.09	4.91	10.03	15.98
AGH-1	4.26	7.01	9.31	2.41	6.17	9.73
AGH-2	4.44	7.37	9.75	2.52	6.68	10.27
DGH	4.34	6.47	9.27	2.64	5.99	10.23

Table 5: Comparison with deep hashing methods in mAP (%) on CIFAR-10 and NUS-WIDE with 16, 24, 32 and 48 bits. We directly cite the results of DHN, DNNH, CNNH⋆ and CNNH from [51] and the results of SSDH from [49]. Here, the mAP is calculated based on the top 5,000 returned neighbors for NUS-WIDE , and the 4,096-d deep features are used.

Methods	CIFAR-10				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
Ours	**80.4**	**81.3**	**81.5**	**82.4**	**72.2**	**74.8**	**74.9**	**76.1**
SSDH	80.1	81.3	81.2	81.4	70.7	72.5	73.1	73.5
DHN	55.5	59.4	60.3	62.1	70.8	73.5	74.8	75.8
DNNH	55.2	56.6	55.8	58.1	67.4	69.7	71.3	71.5
CNNH⋆	48.4	47.6	47.2	48.9	61.7	66.3	65.7	68.8
CNNH	42.9	51.1	50.9	52.2	61.1	61.8	62.5	60.8
S3PLH	42.0	47.9	49.9	51.6	45.7	44.7	44.9	43.9
SDH	48.1	64.2	66.0	67.1	62.3	62.9	63.6	62.1
KSH	36.9	39.9	43.2	44.0	66.6	68.1	68.3	68.5
FastH	62.7	68.7	70.4	72.3	62.2	64.1	64.7	65.4

4. Model Analysis

In this section, we evaluate the proposed method from different aspects. For this evaluation, we conduct all the experiments on CIFAR-10 with 16-bit codes.

4.1. Parameter Sensitivity

Fig. 5 shows the effects of the parameters λ, μ and γ, respectively. We vary λ, μ and γ in {0, 1e-4, 1e-3, 1e-2, 1e-1, 1}. We focus on the bit uncorrelation and balance constraints first. As we can see clearly, imposing these two constraints improves the performance. In terms of the bit uncorrelation constraint, a 3% improvement in mAP is obtained when λ = 1e-3, and the mAP decreases as λ increases. For the bit balance constraint, the mAP deteriorates rapidly as with the increasing μ. The parameter γ controls the quantization error between binary codes and auxiliary variables. As can be seen from Fig. 5, the network is not sensitive to γ.

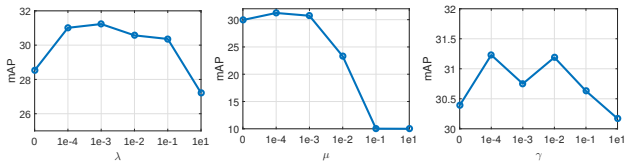

Figure 5: The results of mAP w.r.t. λ, μ and γ, respectively. The evaluation is performed on CIFAR-10 with 16-bit codes.

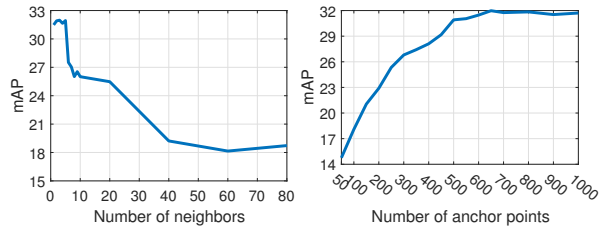

Figure 6: The results of mAP w.r.t. the number of neighbors (left) and that of anchors (right). The evaluation is performed on CIFAR-10 with 16-bit codes.

Table 6: The influence of different proportions of the labeled/unlabeled data in a batch.

Proportion	20/100	40/80	60/60	80/40	100/20
mAP (%)	31.61	31.98	26.59	24.76	22.11
Precision@500 (%)	56.03	56.21	45.13	43.71	40.06

4.2. Graph Convolutional Layers

We also investigate into the (asymmetric) graph convolutional layers, in which the (asymmetric) adjacency matrix is a key component. Since the matrix is defined by the prechosen anchor points, we evaluate the effects of different numbers of anchor points. As Fig. 6 illustrates, the mAP increases from 15% to 32% with the increasing number of anchor points. Using over 650 anchor points, the mAP reaches saturation and stays around 32%.

In the asymmetric graph convolution layer, the number of nearest anchor points (*i.e.*, s) plays an important role in constructing the asymmetric adjacency matrix. Here, we show our results with different numbers of nearest anchor points (neighbors) in Fig. 6. From the figure, we can see that mAP consistently drops with the increasing number of neighbors. When the number of neighbors is between 1 and 7, mAP keeps around 31%, and then there is a sharp decrease with more than 8 nearest neighbors.

4.3. Proportion of Labeled Data to Unlabeled Data

In our experiments, we empirically use 40 labeled and 80 unlabeled data points in a batch. We report the performance of different proportions of labeled/unlabeled data in Table 6. With the increasing number of labeled points in a batch, the performance deteriorates rapidly. This phenomenon is explainable: Since there are more unlabeled samples than labeled ones in the training set, exceeding the proportion of labeled data in a batch will make the network too dependent on the labeled data, and thus become over-fitting.

Discussion. In our experiments, we use the simple four-layer network architecture for GCNH, which is composed of one GC layer, two AGC layers, and one fully connected layer. We first investigate into the influence of the model depth on the retrieval performance. Here, we only increase the number of GC layers. As illustrated in Fig. 7, mAP increases slightly with more layers but drops rapidly from 4 layers on. This phenomenon is common when training deeper networks: as the network depth increasing, the accuracy becomes saturated and then degrades rapidly [14]. Recently, many efforts have been devoted to addressing this problem. Identity mapping [14, 43] provides a solution, in which the subsequent layers directly copy the outputs of the preceding layers. Inspired by [14], we train a deeper model by applying residual connections (identity mapping) between the GC layers:

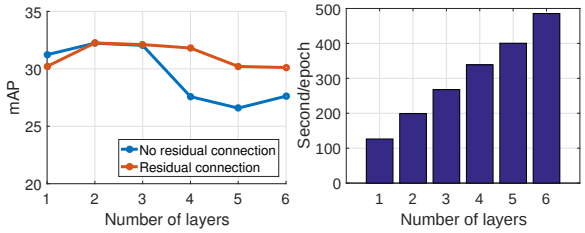

Figure 7: (Left) Influence of model depth (number of graph convolutional layers) on retrieval performance and (right) the training time per epoch of different numbers of layers (graph convolutional layers). The evaluation is performed on CIFAR-10 with 16-bit codes.

$$\mathbf{H}^{(l)} = \sigma\left(\widetilde{\mathbf{Z}}\mathbf{H}^{(l)}\mathbf{W}^{(l-1)}\right) + \mathbf{H}^{(l-1)}, \qquad (13)$$

in which σ denotes the ReLU activation function. As shown in Fig. 7, the residual connections enable the model to achieve more stable performance with increasing depths. We also report the training time of our model in Fig. 7, from which we can see that the training time grows linearly with the increasing number of GC layers.

5. Conclusion

In this paper, we facilitated the graph convolutional networks (GCNs) with an additional ability of learning semantics/similarity-preserving binary embeddings for massive image datasets. To address the scalability and out-of-sample extension issues of GCNs, we proposed a novel asymmetric graph convolutional layer leveraging data anchors, which was effectively stacked into a graph convolutional network based hashing (GCNH) framework. The GCNH framework was successfully applied to tackle the semi-supervised hashing problem. The extensive experiments corroborate that GCNH significantly outperforms the state-of-the-art hashing methods on large-scale image benchmarks.

References

[1] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint*, arXiv:1312.6203, 2013. 2

[2] F. Cakir, K. He, S. Adel Bargal, and S. Sclaroff. Mihash: Online hashing with mutual information. In *Proc. ICCV*, 2017. 1

[3] Z. Cao, M. Long, J. Wang, and P. S. Yu. HashNet: Deep learning to hash by continuation. In *Proc. ICCV*, 2017. 1

[4] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng. NUS-WIDE: A real-world web image database from national university of singapore. In *Proc. ACM CIVR*, 2009. 5

[5] C. Da, S. Xu, K. Ding, G. Meng, S. Xiang, and C. Pan. Amvh: Asymmetric multi-valued hashing. In *Proc. CVPR*, 2017. 1

[6] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. NIPS*, 2016. 2

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. CVPR*, 2009. 5

[8] T.-T. Do, A.-D. Doan, and N.-M. Cheung. Learning to hash with binary deep neural network. In *Proc. ECCV*, 2016. 1

[9] T.-T. Do, D.-K. Le Tan, T. T. Pham, and N.-M. Cheung. Simultaneous feature aggregating and hashing for large-scale image search. In *Proc. CVPR*, 2017. 1

[10] V. Erin Liong, J. Lu, Y.-P. Tan, and J. Zhou. Cross-modal deep variational hashing. In *Proc. ICCV*, 2017. 1

[11] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. VLDB*, 1999. 1

[12] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proc. CVPR*, 2011. 1, 6

[13] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. In *Applied and Computational Harmonic Analysis*, 2011. 2

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016. 8

[15] H. Jain, J. Zepeda, P. Perez, and R. Gribonval. Subic: A supervised, structured binary code for image search. In *Proc. ICCV*, 2017. 1

[16] Q.-Y. Jiang and W.-J. Li. Scalable graph hashing with feature transformation. In *Proc. IJCAI*, 2015. 1

[17] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2, 3

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*. 2012. 1, 7

[19] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proc. ICCV*, 2009. 1

[20] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE TPAMI*, 31(12), 2009. 1

[21] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proc. CVPR*, 2015. 1, 6

[22] W.-J. Li, S. Wang, and W.-C. Kang. Feature learning based deep supervised hashing with pairwise labels. In *Proc. IJCAI*, 2016. 1

[23] X. Li, D. Hu, and F. Nie. Large graph hashing with spectral rotation. In *Proc. AAAI*, 2017. 1

[24] X. Li, C. Shen, A. Dick, and A. van den Hengel. Learning compact binary codes for visual tracking. In *Proc. CVPR*, 2013. 1

[25] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel. Gated graph sequence neural networks. *CoRR*, abs/1511.05493, 2015. 2

[26] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Proc. CVPR*, 2014. 1, 6

[27] G. Lin, C. Shen, D. Suter, and A. v. d. Hengel. A general two-step approach to learning-based hashing. In *Proc. ICCV*, 2013. 1

[28] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *Proc. CVPR*, 2015. 1

[29] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *Proc. CVPR*, 2016. 1

[30] H. Liu, R. Wang, S. Shan, and X. Chen. Learning multifunctional binary codes for both category and attribute oriented retrieval tasks. In *Proc. CVPR*, 2017. 1

[31] L. Liu, L. Shao, F. Shen, and M. Yu. Discretely coding semantic rank orders for image hashing. In *Proc. CVPR*, 2017. 1

[32] W. Liu, J. He, and S.-F. Chang. Large graph construction for scalable semi-supervised learning. In *Proc. ICML*, 2010. 1, 2

[33] W. Liu, C. Mu, S. Kumar, and S.-F. Chang. Discrete graph hashing. In *Proc. NIPS*, 2014. 1, 2, 6

[34] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proc. CVPR*, 2012. 1, 5, 6

[35] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proc. ICML*, 2011. 1, 6

[36] Y. Mu, G. Hua, W. Fan, and S.-F. Chang. Hash-svm: Scalable kernel machines for large-scale visual classification. In *Proc. CVPR*, 2014. 1

[37] J. Qin, L. Liu, L. Shao, B. Ni, C. Chen, F. Shen, and Y. Wang. Binary coding for partial action analysis with limited observation ratios. In *Proc. CVPR*, 2017. 1

[38] J. Qin, L. Liu, L. Shao, F. Shen, B. Ni, J. Chen, and Y. Wang. Zero-shot action recognition with error-correcting output codes. In *Pro. CVPR*, 2017. 1

[39] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *Proc. CVPR*, 2015. 1, 6

[40] F. Shen, C. Shen, Q. Shi, A. van den Hengel, and Z. Tang. Inductive hashing on manifolds. In *Proc. CVPR*, 2013. 1

[41] A. Shrivastava and P. Li. Asymmetric lsh (ALSH) for sublinear time maximum inner product search (MIPS). In *Proc. NIPS*, 2014. 1

[42] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015. 1, 6

[43] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. In *Proc. ICML*, 2015. 8

[44] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *TPAMI*, 34(1), 2012. 1

[45] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. CVPR*, 2015. 1

[46] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large scale search. *IEEE TPAMI*, 34(12), 2012. 1, 6

[47] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. NIPS*, 2008. 1

[48] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *Proc. AAAI*. 1, 6, 7

[49] J. Zhang and Y. Peng. SSDH: semi-supervised deep hashing for large scale image retrieval. *IEEE TCSVT*. 6, 7

[50] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proc. CVPR*, 2015. 1

[51] H. Zhu, M. Long, J. Wang, and Y. Cao. Deep hashing network for efficient similarity retrieval. In *Proc. AAAI*, 2016. 6, 7

[52] B. Zhuang, G. Lin, C. Shen, and I. Reid. Fast training of triplet-based deep binary embedding networks. In *Proc. CVPR*, 2016. 1